C Boolean

In C, Boolean is a data type that contains two types of values, i.e., 0 and 1. Basically, the bool type value represents two types of behavior, either true or false. Here, '0' represents false value, while '1' represents true value.

In C Boolean, '0' is stored as 0, and another integer is stored as 1. We do not require to use any header file to use the Boolean data type in [C++](https://www.javatpoint.com/cpp-tutorial), but in C, we have to use the header file, i.e., stdbool.h. If we do not use the header file, then the program will not compile.

Syntax

1. **bool** variable\_name;

In the above syntax, **bool** is the data type of the variable, and **variable\_name** is the name of the variable.

**Let's understand through an example.**

1. #include <stdio.h>
2. #include<stdbool.h>
3. **int** main()
4. {
5. **bool** x=**false**; // variable initialization.
6. **if**(x==**true**) // conditional statements
7. {
8. printf("The value of x is true");
9. }
10. **else**
11. printf("The value of x is FALSE");
12. **return** 0;
13. }

In the above code, we have used **<stdbool.h>** header file so that we can use the bool type variable in our program. After the declaration of the header file, we create the bool type variable '**x**' and assigns a '**false**' value to it. Then, we add the conditional statements, i.e., **if..else**, to determine whether the value of 'x' is true or not.

**Output**

The value of x is FALSE

Boolean Array

Now, we create a bool type array. The Boolean array can contain either true or false value, and the values of the array can be accessed with the help of indexing.

**Let's understand this scenario through an example.**

1. #include <stdio.h>
2. #include<stdbool.h>
3. **int** main()
4. {
5. **bool** b[2]={**true**,**false**}; // Boolean type array
6. **for**(**int** i=0;i<2;i++) // for loop
7. {
8. printf("%d,",b[i]); // printf statement
9. }
10. **return** 0;
11. }

In the above code, we have declared a Boolean type array containing two values, i.e., true and false.

**Output**

1,0,

typedef

There is another way of using Boolean value, i.e., **typedef**. Basically, typedef is a [keyword in C language](https://www.javatpoint.com/keywords-in-c), which is used to assign the name to the already existing datatype.

**Let's see a simple example of typedef.**

1. #include <stdio.h>
2. **typedef** **enum**{**false**,**true**} b;
3. **int** main()
4. {
5. b x=**false**; // variable initialization
6. **if**(x==**true**) // conditional statements
7. {
8. printf("The value of x is true");
9. }
10. **else**
11. {
12. printf("The value of x is false");
13. }
14. **return** 0;
15. }

In the above code, we use the Boolean values, i.e., true and false, but we have not used the bool type. We use the Boolean values by creating a new name of the 'bool' type. In order to achieve this, **the typedef** keyword is used in the program.

1. **typedef** **enum**{**false**,**true**} b;

The above statement creates a new name for the '**bool**' type, i.e., 'b' as 'b' can contain either true or false value. We use the 'b' type in our program and create the 'x' variable of type 'b'.

**Output**

The value of x is false

Boolean with Logical Operators

The Boolean type value is associated with logical operators. There are three types of logical operators in the [C language](https://www.javatpoint.com/c-programming-language-tutorial):

**&&(AND Operator):** It is a logical operator that takes two operands. If the value of both the operands are true, then this operator returns true otherwise false

**||(OR Operator):** It is a logical operator that takes two operands. If the value of both the operands is false, then it returns false otherwise true.

**!(NOT Operator):** It is a NOT operator that takes one operand. If the value of the operand is false, then it returns true, and if the value of the operand is true, then it returns false.

**Let's understand through an example.**

1. #include <stdio.h>
2. #include<stdbool.h>
3. **int** main()
4. {
5. **bool** x=**false**;
6. **bool** y=**true**;
7. printf("The value of x&&y is %d", x&&y);
8. printf("\nThe value of x||y is %d", x||y);
9. printf("\nThe value of !x is %d", !x);
10. }

**Output**

The value of x&&y is 0

The value of x||y is 1

The value of !x is 1